

APPARATUS AND METHOD FOR DETECTING MEMORY LEAKS

BACKGROUND

1. TECHNICAL FIELD

The invention relates generally to computing systems, and more particularly to
5 a system and method for detecting memory leaks in applications running on
monitored computer systems.

2. DESCRIPTION OF THE RELATED ART

A memory “leak” occurs when an application allocates a block of memory
(either directly or indirectly through the use of various programming interfaces), and
10 then fails to deallocate the memory when it is no longer required. Memory leaks
consume system resources that could otherwise be available to the operating system
and other applications.

Failure to release allocated memory is a very common programming error.
Due to the complexity of software applications and their memory management, such
15 memory leaks frequently exist in software delivered to end users. Because the
memory requirements of applications change dynamically over time, it can be very
difficult for an end-user to detect memory leaks in software that they are using.

When memory leaks go undetected, system resources that they consume
become unavailable to the operating system and other applications. This has the
20 effect of reducing performance, and in some cases, causing unscheduled downtime for
applications and the operating system.

There are many memory leak detection solutions available to application
developers. Such conventional solutions, however, require that the application be
instrumented for memory leak detection through source code modifications, the use of
25 an application debugger, or through libraries linked to applications. While these
known solutions can be fairly precise in identifying memory leaks and can often
pinpoint the source code where the leaked memory was allocated, they have the
shortcoming that they require access to or modifications of the application in order to
perform the memory leak detection. Such known solutions therefore do not solve the

problem faced by typical end users that do not have access to the application source code. Additionally, end users wish to monitor applications in their normal mode of operation not in a special “debug” mode.

5 There are also known solutions available that can report when a system’s resources have been consumed to the point that performance is suffering. These known solutions, however, are unable to determine which application is leaking. Additionally, they often report the problem when it has reached a crisis level. Finally, they may be unable to detect small memory leaks at all.

10 There is therefore a need for a method and system for detecting memory leaks that minimizes and/or eliminates one or more of the shortcomings set forth above.

SUMMARY OF THE INVENTION

One object of the invention is to provide a solution to one or more of the shortcomings set forth in the Background. The present invention has several advantages. One advantage of the present invention is that it does not require access
15 to and modification of the source code of the monitored application. Another advantage is that it does not have to be instrumented, does not require an application debugger nor does it require linkage to a special set of library routines. Another advantage is that it is operative to detect memory leaks while the monitored application runs in a normal mode, not a special debug mode. Finally, the present
20 invention provides detection of memory leaks long before a crisis level of resource depletion has occurred.

These and other objects, features and advantages of the present invention may be achieved through a method and apparatus for detecting memory leaks for an application program executing on a computer. The method, in one embodiment,
25 includes the step of determining when a peak allocated memory level has increased a determined number of times during a determined time interval. In a preferred embodiment, the method includes the further step of filtering out increases in peak allocated memory levels that are not indicative of a memory leak associated with the application program. The filtering step may include the substep of ignoring increases
30 in peak allocated memory levels during a startup time interval immediately after the

monitored application program begins to execute. The filtering step may further include the substep of ignoring increases in peak allocated memory levels that occur after the startup time interval but that occur less than a preselected time apart, which are not necessarily indicative of a memory leak and may be indicative of normal
5 memory allocation activity.

An apparatus for detecting memory leaks is also presented.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to the drawings wherein like reference numerals are used to identify identical components in the various views, FIG. 1 shows a computer 10
10 having a memory leak detection and reporting system 12 according to the invention. The system 12 includes (i) a monitoring agent ("agent") 14 for monitoring peak allocated memory levels at a predetermined sampling frequency; (ii) a user interface 16 for configuring operating parameters of system 12; and (iii) a reporting mechanism 18 for logging events, generating notifications and/or taking prescribed
15 actions. FIG. 1 shows a stand-alone embodiment wherein system 12 executes on the same computer 10 as a monitored application 20. Monitored application 20 has an associated allocated memory level in a memory 22. It should be understood that the monitored application may employ or use memory from one or more memory pools (if supported by the operating system), and the present invention is configured to be
20 compatible with such an arrangement. In one embodiment, agent 12 is provided as application program, but could alternately be incorporated in an operating system, such as O/S 24, of computer 10 or alternatively implemented using hardware components.

FIG. 2 shows an alternate, distributed computing embodiment of the present
25 invention operating over a network. For example, where computer 10 is a first computer, FIG. 2 shows system 12 on the first computer 10 configured to detect a memory leak on one or more of the monitored applications $20_1, 20_2, \dots 20_n$ executing on other computers. The first and second (other) computers are in communication with each other over a network 24. In this alternate embodiment, agent 14 of
30 system 12 collects data on the monitored application 20 over network 24. Data concerning the monitored applications may be stored in a local database (not shown)

available to system 12. Network 24 may comprise any kind of communication network now known or hereafter developed, including without limitation a local area network, a wide area network, the Internet, or network. Monitored applications 20₁, 20₂, . . . 20_n may be server applications, for example, with computer 10 being a
5 separate system/display where reporting may be made.

In addition, computer 10 may comprise any conventional computing apparatus, and, in one embodiment, may comprise a Windows-based computer. It should be understood, however, that the present invention is not so limited. Computer 10 may be based on other widely available operating systems 24, such as,
10 but not limited to, Unix-based systems, Linux-based systems, and Apple Macintosh-based systems.

Before proceeding to a detailed description of the system and method of the present invention, an brief description will be set forth of several characteristics of memory allocation generally, and the memory leak mechanism specifically, that
15 Applicant has discovered. The purpose of this section will become apparent as this detailed description proceeds.

FIG. 3 is a simplified timing diagram showing an exemplary allocated memory versus time graph. The level of the allocated memory is shown by trace 26 and is shown on a per application basis. Note at time zero, immediately prior to the
20 beginning execution of the monitored application 20, the allocated memory is zero.

In general, the present invention monitors the allocated memory levels at a predetermined frequency, and determines peak allocated memory levels. The invention is further configured to determine increases in the peak allocated memory levels. The present invention determines that a memory leak exists when it has
25 counted a predetermined number of increases in the peak allocated memory level. The predetermined number of times is herein sometimes referred to as an Alarm Limit. The predetermined number of times may, in one embodiment, comprise a determined number of times, being either (1) fixed in advance of the operation of the inventive program (*e.g.*, user selected), or (2) dynamically set by the system during
30 operation based on criteria. Exemplary peaks that represent an increase over the prior

“peak” level indicative of memory leaks are designated P_1 , P_2 , P_3 , and P_4 in FIG. 3. Note that some increases in peak allocated memory levels do not carry the P_i designation. These will be explained in turn.

First, there is the startup period. The time interval from time zero to the time designated t_1 is herein referred to as a startup time interval 28. This is shown in FIG. 3 enclosed in a dashed-line box. Applications frequently allocate memory that is used only for initialization during their startup period. Such memory usage often creates peaks that are not indicative of a memory leak. Employing the invention during startup might also set a “high” peak that would obscure later peaks that occur as an application really leaks memory. In accordance with the present invention, increases in peak allocated memory levels are filtered out or ignored during startup period 28 as not being indicative of a memory leak. These do not count toward the predetermined number of times (Alarm Limit) required to conclude that there is a memory leak in the monitored application 20.

Second, there are normal “burst” increases in allocated memory. The present invention uses a time delay between successive “peaks” to filter out (ignore) certain increases in the peak allocated memory level. When two increases occur too close together the latter increase is ignored and is not counted toward the Alarm Limit. The time delay is used to avoid bursts of memory allocation activity that may occur in normal processing from rapidly counting toward the Alarm Limit, thereby triggering, perhaps falsely, the presence of a memory leak. While such rapid increases may be ambiguous as to whether there is in actuality a memory leak or not, real memory leaks that are significant tend to occur many times over a long period of time. Thus, ignoring closely spaced increases removes the ambiguity and improves the reliability of the detection. Filtering out increases in peak allocated memory of this type help minimize or eliminate false alarms. For example, the successive increase in peak allocated memory occurring at time point t_4 is not necessarily indicative of a memory leak and may simply reflect normal processing (*i.e.*, it occurs too closely in time to the prior peak at time t_3). This increase in peak allocated memory is therefore filtered out (ignored) in counting towards the Alarm Limit. This peak at t_4 , along with the peak at t_3 , are enclosed in dashed-line box designated 30.

FIG. 4 is a simplified block diagram showing, in greater detail, system 12 of FIG. 1 and FIG. 2. FIG. 4 shows monitoring agent 14, interface 16, reporting mechanism 18, and a plurality of operating parameters 32₁ through 32₉. Interface 16 is configured so as to allow a user of the inventive system 12 to specify a particular value or contents for at least one of, and preferably all, of the operating parameters 32₁ through 32₉. These operating parameters control the operation of the inventive method, and whose values or contents may be stored locally in a file, although such values could be stored, in alternative embodiments, in database or even hard-coded. In one embodiment, one set of values for the operating parameters is used. It should be understood, however, that in alternative embodiments, different sets of values for the operating parameters may be defined for different, respective monitored environments.

In one embodiment, the operating parameters include:

1. DETECTION_ENABLED (type Boolean). This parameter is used to globally enable and disable the operation of the memory leak detection system.
2. SAMPLING_RATE (type integer). This parameter controls the frequency at which the memory usage of monitored applications is sampled.
3. STARTUP_PERIOD (type integer). This parameter specifies the time, in seconds, that is considered part of startup period for monitored applications.
4. ALARM_LIMIT (type integer). This parameter specifies the number of increases in the peak allocated memory size that must occur before an application is flagged as leaking memory.
5. TIME_DELAY (type integer). This parameter specifies the minimum time, in seconds, between increases in peak allocated memory size that count toward the application reaching ALARM_LIMIT.
6. ALARM_ACTIONS. This parameter specifies what actions, if any should be taken when a memory leak alarm is triggered.
7. ALARM_NOTIFICATIONS. This parameter specifies what notification mechanisms should be used to inform the user when a memory leak alarm is triggered.
8. MINIMUM_RATE_FOR_ALARM (type floating point). This parameter specifies the minimum leakage rate, in kilobytes per minute, that triggers an alarm.

9. IGNORED_APPLICATIONS (type array of strings). This parameter specifies a list of application on which memory leak detection is not performed.

In one embodiment, default values may be as follows: Sampling Rate (5 seconds); Startup Period (180 seconds); Alarm Limit (45 times); Time Delay (75 seconds); and the Minimum Rate For Alarm (200 bytes/minute). Of course, as indicated, these default values may be changed by a user by way of the interface 16.

FIG. 4 also shows a reporting mechanism 18. In one embodiment, the reporting mechanism may include an alarm database 19 for allowing an alarm interface to record an alarm/response for later retrieval/review, a log file, an interface to the system event log, an e-mail interface for specifying one or more (*i.e.*, a list) of e-mail addresses to which notifications are made, an Simple Network Monitoring Protocol (SNMP) interface, as well as other notification mechanisms. Reporting mechanism 18 is configured to allow selection of zero or more notifications from the group, to be performed upon detection of memory leak in excess of the minimum rate for alarm, comprising an e-mail message, a simple network monitoring protocol (SNMP) message, a telecommunications page, a visual notification on a display associated with a computer, an audio notification, and a combination of any of the foregoing notifications.

The reporting mechanism 18 is also configured to allow the user to select an action (to be performed when a memory leak exceeds the minimum rate for alarm) from the group comprising no action, running a program or script, killing the program, rebooting the computer on which the program is executing and a combination of any of the foregoing actions. Of course, other responses, either in the nature of a notification or an action, may be included and remain within the spirit and scope of the present invention.

FIG. 5 is a simplified flow chart showing the operation of a method in accordance with the present invention. The method begins with step 34.

In step 34, the execution of the present invention begins. In one embodiment, a mechanism is provided that detects the start of new applications. Such mechanisms are known for performing this function. The remainder of the description is

performed independently on each application that is monitored for memory leaks by system 12. When a new application is started, it is added to a list (not illustrated) of tracked applications. Tracked applications are removed from this list upon termination.

5 In step 34, generally, agent 14 monitors a tracked or monitored application to determine allocated memory usage. Step 34 may include the substeps of sampling the memory level at the frequency determined by `SAMPLING_RATE` operating parameter 32₂ for each monitored application for which a memory leak has not
10 determine the current total allocated memory for the monitored application. The determined memory level is recorded (unless it occurs during the startup period, which is explained in greater detail below in connection with the filtering step 36). To the extent that a new “peak” has not been reached, the method may, in an alternate embodiment, proceed to the next monitored application on the list. However, if a new
15 “peak” is determined to have been achieved, then the time of the new peak allocated memory is also recorded for later use. It should be understood that other embodiments could use other mechanisms to determine if an application should or should not be monitored for memory leaks. In addition, it should be further understood that other mechanisms could be used to determine the sampling times,
20 including event-based mechanisms triggered by application memory allocation requests. The method proceeds to step 36.

 In step 36, the method is configured for filtering out increases in peak allocated memory levels that are not necessarily indicative of a memory leak. This step may be performed by one or more substeps. For example, step 36 may involve
25 the substep of ignoring increases in peak allocated memory levels during a startup time interval immediately after the monitored program begins to execute. Step 36 may also involve the substep of ignoring increases in peak allocated memory levels that occur after the startup time interval but that occur less than a preselected time apart which are not necessarily indicative of a memory leak and may simply be
30 indicative of a normal burst of memory allocation activities. As to the step of ignoring startup activity, this may be implemented by determining how long the

application has been running, and if it is still in startup as defined by the STARTUP_PERIOD operating parameter 32₃, ignore sampled memory levels and move on to the next monitored application in the list. As to the step of ignoring “burst activity”, this step may be implemented by determining how long it has been since a peak memory increase has been counted toward the ALARM_LIMIT operating parameter 32₄, and, if it has been less than the TIME_DELAY operating parameter 32₅, then ignore the sampled memory levels and move on to the next monitored application. The method then proceeds to step 38.

In step 38, the method determines whether the number of increases in the peak allocated memory has reached a predetermined number of times (*i.e.*, the Alarm Limit). If the answer is NO, then the method branches to the next monitored application on the list, or, in the case of the monitored application the subject of the flow chart in FIG. 4, back up to step 34. If the answer is YES, the method branches to step 40.

In step 40, the method calculates an estimated memory leakage rate. The method may perform this step by determining the difference (*e.g.*, in kilobytes) between the current peak memory level and the initial peak memory level increase after the startup period or the second increase since time zero. Note, the first sampled memory level after the startup period will indicate an “increase” over the initial allocated memory when the program began (which would be zero). This may be considered the first peak increase. What is being referred to here as the initial increase after startup is the next increase in the peak level or the second increase in the peak level. In any event, the difference (*e.g.*, in minutes) is also determined between the current peak memory level and the second peak memory level (as defined above). Dividing the difference in the memory level by the difference in time will yield a memory leakage rate. Of course, alternative mathematical approaches for calculating a memory leakage rate may be used and remain within the spirit and scope of the present invention. The method proceeds to step 42.

In step 42, the method compares and determines whether the calculated memory leakage rate exceeds the minimum rate for alarm operating parameter 32₈. If the answer is NO, then the method branches to the next monitored application on the

list, or, in the case of the monitored application at hand, back up to step 34. If the answer is YES, and, optionally, the monitored application is not in the IGNORED_APPLICATIONS list (operating parameter 32, set forth above) then the method branches to step 44.

- 5 In step 44, the method produces a response in the form of either a notification or an action, as described in detail above with respect to reporting mechanism 18. The method proceeds to step 46.

 In step 46, the method records the response (notification, action or both) for later review.

- 10 FIG. 6 is a simplified flowchart showing, in greater detail, the alarm response step 44 in Figure 5. This detailed flowchart starts with step 44, which assumes a response (action or notification) has been specified by the user, and then proceeds to a decision step 48.

- In step 48, the method determines whether the specified response(s) is an
15 action. If the answer to decision block 48 is YES, then the method branches to step 52. In step 52, the method selects one or more of the specified actions and executes them. These actions may include “no action”, running a program or script for performing additional functions, killing (terminating) the application program with the memory leak, rebooting the computer (*e.g.*, rebooting a server if the computer
20 is a configured as a server), or any combination of the above. The method then proceeds to step 49.

 Step 49 also accepts the flow of control when the answer in the decision block 48 is “NO.” In step 49, the method determines whether the specified response(s) include a notification. If “YES,” then the method branches to step 50.

- 25 In step 50, the method selects one or more of the specified notifications and executes them. These may include sending a notification as to the alarm in the form of an e-message to one or more e-mail addresses specified in a list, an SNMP message, a telecommunications page, a visual or audio message, or any combination of the above.

Upon completion of step 50, the method branches to step 46, described above.

If the answer to decision block 49 is "NO," then the method branches to an "end" block of this subpart of the method shown in Figure 6.

It should be understood that the functions and methodologies may be performed through appropriate programming of computer 10 using conventional programming tools known to those of ordinary skill in the art. Accordingly, the computer 10 configured by way of programming constitute the structure corresponding to the recited functions in the apparatus claims. Other structures, however, are contemplated including without limitation electronic hardware, including computer hardware suitably configured to perform the functions recited in the claims.

In support thereof of the foregoing, pseudocode for implementing an exemplary embodiment is hereby reproduced below.

```

15  at SAMPLING_RATE frequency:
    for each "app" that is active:

        // Ignore the process if we have an active memory leak alarm or if
        // the
        // process is still in its startup period
20  if ( (DETECTION_ENABLED)
        && (!MemLeakAlarmIsActive)
        && (TimeLoaded(app) > STARTUP_PERIOD))
    {
        // Has the VM size increased?
25  if (CurrentPeakMemoryUsage > RecordedPeakMemoryUsage)
    {
        RecordedPeakMemoryUsage = CurrentPeakMemoryUsage;

        // Has the inter-increase delay period expired since the last
        // peak VM increase that we measured?
30  if ((GetCurrentTime() - LastIncTime) > TIME_DELAY)
    {
        // Count this peak VM size increase
        LastIncTime = GetCurrentTime();
35  NumVmIncreases++;

        // Record usage a second increment so that we can later
        // estimate a rate
40  if (NumVmIncreases == 2)
    {
        UsageAtSecondIncTime = CurrentPeakMemoryUsage;
        SecondIncTime         = LastIncTime;
    }
    }
    }

```

```

// If we have reached the configured limit, signal an
alarm.
5      if (NumVmIncreases >= ALARM_LIMIT)
      {
          // Estimate the leakage rate in kilobytes per minute
          double leakageRate = ((CurrentPeakMemoryUsage -
10      UsageAtSecondIncTime)
                                / 1000.0)
                                / ((LastIncTime - SecondIncTime)
                                / 60.0);

15      // Ignore leaks that are smaller than the configured
          // alarming threshold
          if (leakageRate > MINIMUM_RATE_FOR_ALARM)
          {
20      // Make sure that this app is not in the "ignore"
          // list for memory leaks
          if (!AppIsIgnored (app))
          {
25      // call the alarm class to start a new alarm,
          // record it in the database, take actions and
          // perform notifications
          StartAlarm ()
          MemLeakAlarmIsActive = true
          }
          }
30      }
      }
      }
      }

```

35 It is to be understood that the above description is merely exemplary rather than limiting in nature, the invention being limited only by the appended claims. Various modifications and changes may be made thereto by one of ordinary skill in the art which embody the principles of the invention and fall within the spirit and scope thereof.